Resource-Constrained Classification Using a Cascade of Neural Network Layers

Sam Leroux, Steven Bohez, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, Bart Dhoedt
Ghent University - iMinds
Gaston Crommenlaan 8/201
B-9050 Ghent, Belgium
{sam.leroux, steven.bohez, tim.verbelen, bert.vankeirsbilck,
pieter.simoens, bart.dhoedt }@intec.ugent.be

***Abstract*—Deep neural networks are the state of the art technique for a wide variety of classification problems. Although deeper networks are able to make more accurate classifications, the value brought by an additional hidden layer diminishes rapidly. Even shallow networks are able to achieve relatively good results on various classification problems. Only for a small subset of the samples do the deeper layers make a significant difference. We describe an architecture in which only the samples that can not be classified with a sufficient confidence by a shallow network have to be processed by the deeper layers. Instead of training a network with one output layer at the end of the network, we train several output layers, one for each hidden layer. When an output layer is sufficiently confident in this result, we stop propagating at this layer and the deeper layers need not be evaluated. The choice of a threshold confidence value allows us to trade-off accuracy and speed.**

Applied in the Internet-of-things (IoT) context, this approach makes it possible to distribute the layers of a neural network between low powered devices and powerful servers in the cloud. We only need the remote layers when the local layers are unable to make an accurate classification. Such an architecture adds the intelligence of a deep neural network to resource constrained devices such as sensor nodes and various IoT devices.

We evaluated our approach on the MNIST and CIFAR10 datasets. On the MNIST dataset, we retain the same accuracy at half the computational cost. On the more difficult CIFAR10 dataset we were able to obtain a relative speed-up of 33% at an marginal increase in error rate from 15.3% to 15.8%.

I. Introduction

While extremely powerful, deep neural networks are also resource demanding. At training time this problem is often addressed by the use of a Graphics Processing Unit (GPU), which are better suited for the calculations required when training a neural network because of their parallel nature. GPUs now make it possible to train networks that were previously considered to be too difficult to train. Efficient GPU implementations are often put forward as one of the main reasons why deep neural networks have become so successful [1].

Training the deep network is undoubtedly the most computationally expensive task. But once trained, these networks have to be deployed in real-world environments. Current research directions indeed investigate the use of deep neural networks in smartphones, sensor networks, robots, drones and various IoT applications [2]. In these environments, considerations of weight and size impose intrinsic resource limitations on processor power and battery capacity. Although the training step is the computationally most challenging task, the evaluation of a trained network is also demanding, especially on these constrained devices typical for IoT applications.

In this paper we propose a methodology of speeding up the evaluation of a trained deep neural network. Using our methodology, it is possible to train a deep neural network and to use the trained layers as stages in a cascade. Each stage is able to make more complex decisions but also requires additional computing time. This makes it possible to trade-off accuracy for speed. Every stage in the cascade is able to output a confidence measure. When a certain threshold confidence value is reached, no deeper layers need to be evaluated.

This approach makes it possible to distribute layers across different devices. We can devise an architecture where a deep neural network is distributed between a low power device such as a small robot or sensor node and a high performance server in the cloud. The network may even be too large to deploy completely on the robot alone. When we evaluate all layers of the network in the cloud, we introduce a costly latency factor. Using the cascade architecture, we can evaluate the first layers locally and in most cases obtain a sufficiently confident result using only these layers. Only when the deeper layers are needed, do we have to communicate with the server. Moreover, when the server is not available, the robot is still able to work, although the classifications will be less precise, they are still useful in most cases.

Communicating with a distant server takes time so this should only be done when absolutely necessary. An interesting idea is to bring the cloud closer using Cloudlets [3] which are local representations of the cloud. In essence they provide content and/or processing power accessible via (wireless) LAN. We could extend the previous setup for distributing layers by evaluating intermediate layers on a cloudlet, possibly shared with other robots in the same vicinity. Every stage in the cascade adds captures additional complexity but also introduces an additional latency.

An alternative use case would be to use the output of the local layers to make urgent decisions. The remote layers could give more detailed information afterwards but the details are probably not required for split-second decisions.

An even more interesting case is the use of neuromorphic hardware, computer chips that closely mimic the architecture of the brain [4]. These chips require little power to run so they

TABLE I: Results obtained on the MNIST dataset using feed-forward networks with increasing depths

Network architecture	Test error rate	Reference
1-layer NN (no hidden layer)	12.00%	LeCun et al. [5]
2-layer NN (1 hidden layer)	4.70%	LeCun et al. [5]
3-layer NN (2 hidden layers)	3.05%	LeCun et al. [5]
4-layer NN (3 hidden layers)	1.00%	Salakhutdinov and Hinton [6]
6-layer NN (5 hidden layers)	0.35%	Ciresan et al. [1]

are ideal to add intelligence to robots or sensor devices. Since they are built with the specific purpose of running a neural network, they are much faster than neural networks simulated in software on general purpose hardware. However, these chips typically contain only a relatively small amount of neurons compared to software defined neural networks. An interesting use case would be to equip a robot with such a chip on which the first layer of the network would be evaluated. When the first layer is unable to make a decision with sufficient confidence, a second layer is necessary which could be implemented on the microprocessor of the robot. The deeper layers of the network are then evaluated on a cloudlet or a high performance server in the cloud.

Distributing the layers between devices has an inherent cost: the time needed to serialize and to transfer the data. We focus on constrained devices that are unable to run the entire network. Therefore communication with a cloud back-end is inevitable in this case. Our topology tries to run a part of the network locally. Only the samples that can not be classified with a sufficient confidence have to be processed by the remote system.

In the remainder of this paper we will describe the cascade architecture in more details in section II, as well as the training procedure in section III. Section IV shows the results obtained on two well-known benchmark datasets: MNIST and CIFAR10. Finally in Section V we propose an intuitive technique to select appropriate threshold values, which allows us to trade-off accuracy and speed.

II. Architecture

A typical regular feed-forward neural network consists of an input layer, an output layer and one or more hidden layers in between. The hidden layers transform the output from the previous layer to a more suitable representation for the next layers. Each hidden layer extracts features from its input data while the last layer is a classifier that generates the output (i.e. it classifies the input as belonging to a certain class). Empirical evidence shows that adding more hidden layers makes the network more capable. Table I shows some results achieved on the MNIST [7] dataset by a variety of neural networks with increasing depths. The MNIST dataset consists of 28 by 28 pixel images which show a handwritten digit that need to be classified as the corresponding digit.

The results displayed in table I support the hypothesis that a deep network is able to achieve a higher accuracy than a shallow network. Although single layer neural networks are universal approximators (i.e. they can approximate any continuous function from one finite dimensional space to another with any desired degree of accuracy when a sufficient amount of hidden neurons are available) [8], deep networks

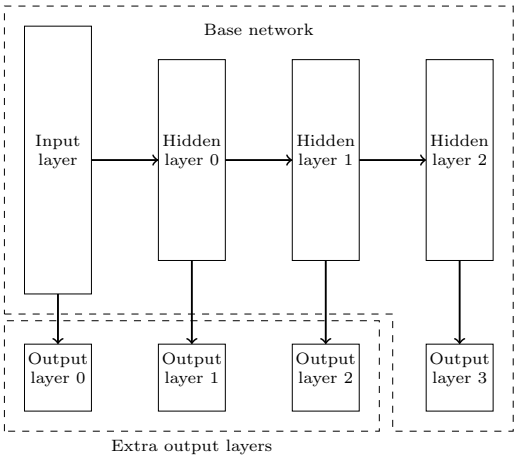


Fig. 1: General architecture of a cascade network

have proven to be much more useful in practice. This is mostly because the deep networks are able to learn a hierarchy of features which enables them to be more efficient in terms of the number of neurons required.

The hypothesis of this paper is that, while deeper networks are indeed able to achieve a higher accuracy, these extra hidden layers are only required for a small subset of samples. The majority of samples can be correctly classified using a shallow network. Table I shows that a network with one hidden layer is able to achieve a 4.7% error rate on the MNIST dataset. This observation suggests that for 95.3% of the samples the one layer network captures enough complexity to classify the samples correctly. For these samples no additional layers are necessary, they only make it more costly to evaluate the network.

We propose an architecture where these redundant calculations can be avoided. Instead of one output layer at the end of the neural network, multiple output layers are trained, one directly attached to the input layer and one after every hidden layer. This makes it possible to stop propagating a sample through the network after an arbitrary layer. To decide at what depth to stop, we rely on the following property of neural network classifiers: the output of a neural network classifier is a good estimate of the posterior probabilities (i.e. how confident is the network that this sample is a member of this class)[9]. When the confidence exceeds a pre-defined threshold, the output from this layer is used as the output of the network. The remaining hidden layers do not have to be evaluated.

Algorithm 1 describes the process. The network consists of n hidden layers and $n+1$ output layers. The first output layer is trained directly on the input data. The other output layers are trained to classify the output after a hidden layer.

Algorithm 1 Propagating a sample through the network

```
procedure FPROP(x)
    i ← 0
    y ← output_layer_i(x)
    while confidence(y) < threshold and i < n do
        x ← hidden_layer_i(x)
        i ← i + 1
        y ← output_layer_i(x)
    return y
```

III. Training

Training the architecture described in the previous sections consists of two separate stages: training the base network and training the additional output layers. The base network consists of the input layer, the hidden layers and the final output layer. The base network is trained in the same way a neural network is traditionally trained, without any special requirements. The hidden layers can be fully connected layers but can also be more complex layers such as convolutional [10] layers.

After the main network is trained, additional output layers (softmax classifiers) can be trained directly on the input data and after each hidden layer. All weights from the base network are fixed and are not allowed to change. This allows the output layers to be trained efficiently. We propagate the training set data once through the network and record the neuron activations. The output layer is then trained to classify the saved activations. Note that at training time no confidence measurements are used, every output layer is trained on the complete training set.

This straight-forward approach to training the cascade allows us to use existing tools to train the network. Additionally, this makes it easy to use pre-trained models as the base network. Only the extra output layers have to be trained which can be done relatively fast.

IV. Results

We experimented with two well known image classification datasets: the MNIST [7] dataset of handwritten digits and the CIFAR-10 [11] dataset containing 32 by 32 pixel color images in ten classes. All experiments described in this paper were performed using the pylearn2 framework [12]. Training was done on an Nvidia GTX780 GPU. The evaluation was done on an Intel Core i5-3340M CPU (2.70GHz).

A. MNIST

The MNIST dataset [7] mentioned before is arguably one of the most common benchmark datasets for classification and image recognition. It consists of a 60,000 sample training set and a 10,000 sample test set. We trained the network shown in Figure 2 to classify the MNIST digits. The network consists of 4 fully connected hidden layers with respectively 800, 1500, 1750 and 2000 neurons. The activations are Rectified Linear Units (ReLU) [13]. We used elastic transformations [14] of the original MNIST training set to generate additional training samples. Dropout [15] allowed us to achieve a test error rate of 0.64% on the base network.

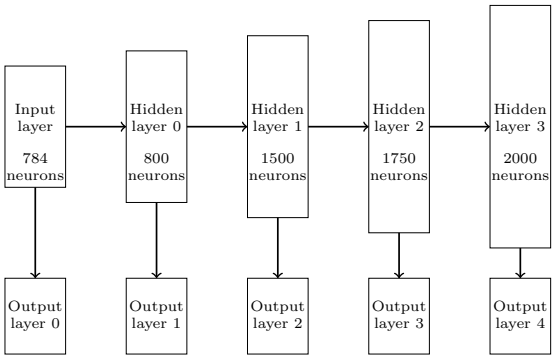


Fig. 2: 4 layer MNIST network

Once the base network is trained, we can train the additional output layers. Training these extra layers is fast compared to training the base network. While the training of the base network takes several hours, training the additional output layers takes only several minutes.

Table II shows the test error and the average time needed to process a test sample for each output layer. While training was done on a GPU, evaluations were performed on CPU as we are mainly interested in applications with resource limitations such as using the network on an IoT-device. All test samples were presented one at a time to the network to simulate an environment where the network has to continually process new information in real-time.

Table II clearly shows that the extra hidden layers have a positive effect on the accuracy, but come at a cost. The time needed to propagate a sample through the four layer network is over twice the time needed for the two layer network. These extra two layers however only make a difference for 0.15% of the samples.

We now use the base network and the extra output layers from the previous sections to build a cascade network. The decisive factor in the cascade is the choice of the threshold value. This value allows a trade-off between accuracy and speed. A low threshold will allow the network to return a less confident result, most likely obtained by an early layer. A high threshold requires a layer to be very confident before returning its result. Table III and Figure 3 show the average accuracy on the MNIST test set and the corresponding runtime with a threshold ranging from 0.9 to 0.999999. We find that a threshold of 0.999 allows for an test error rate of 0.64%, which is the same as the error rate of the base network. Yet the time needed to evaluate the samples is only half that of the base network. The reduction in processing time outweighs the overhead of the extra evaluation of output layers in each step.

TABLE II: Accuracy and runtime at varying depths.

Number of hidden layers	Test error rate	Average time needed to process one test sample (ms)
0	7.61%	0.5
1	1.42%	0.9
2	0.79%	1.5
3	0.69%	2.6
4	0.64%	4.0

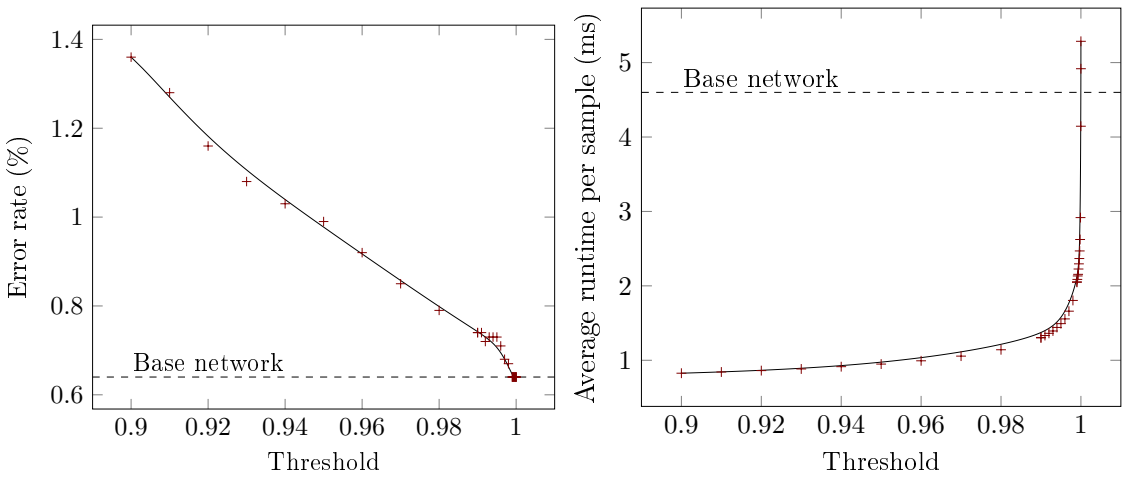


Fig. 3: Error rate (left) and runtime (right) of the cascade using varying thresholds

TABLE III: Accuracy and runtime of the cascade using varying thresholds.

Threshold	Test error rate	Average time needed to process one test sample (ms)
0.9	1.36%	0.8
0.99	0.74%	1.3
0.999	0.64%	2.0
0.9999	0.64%	2.9
0.999 99	0.64%	4.1
0.999 999	0.64%	4.9
1	0.64%	5.2

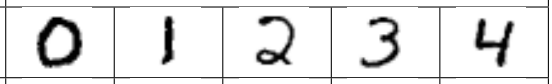
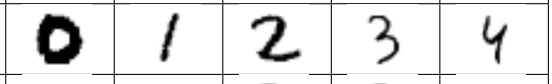
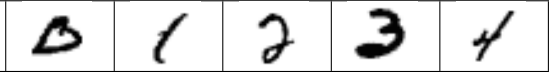
TABLE IV: Percentage of the test samples classified by each layer (threshold = 0.999)

Layer	Classes				
	0	1	2	3	4
0	35.92%	0.09%	21.90%	11.19%	9.78%
1	26.43%	25.55%	24.52%	43.76%	37.68%
2	23.47%	60.35%	27.91%	28.42%	33.60%
3	9.90%	8.11%	17.93%	10.50%	9.37%
4	4.29%	5.90%	7.75%	6.14%	9.57%
	5	6	7	8	9
0	1.35%	19.94%	11.19%	4.11%	0.20%
1	52.24%	33.19%	39.01%	16.22%	4.46%
2	23.65%	29.44%	21.50%	51.44%	59.86%
3	12.33%	11.27%	15.47%	19.30%	23.89%
4	10.43%	6.16%	12.84%	8.93%	11.60%

Layer	Total percentage of samples classified
0	11.48%
1	30.02%
2	36.39%
3	13.78%
4	8.33%

It is interesting to investigate what kinds of samples are classified by each layer. Table IV shows for each class the percentage of the samples of that class being classified by each output layer. Some classes prove to be easier to classify than others, such as the number zero: 35.92% of the samples that represent a zero is classified by the first layer. Surprisingly, the number one proves to be hard to classify, most likely because there are different styles of handwritten ones and because a vertical pen stroke is also a feature of other numbers such as seven and four.

TABLE V: Typical images classified by different layers.

output layer	Typical samples classified by this layer
0	
1	
2	
3	
4	

We assumed that deeper layers are capable of classifying harder samples. Table V shows some typical examples of the images classified by each output layer. This table visually supports our assumption, with the easier samples being classified by the first layers, while the harder samples are left for the deeper layers.

B. *CIFAR10*

The results achieved on the MNIST dataset are promising but the question remains whether these results generalize to more complex datasets and network architectures. In this section we evaluated our approach on the CIFAR10 [11] dataset. The CIFAR10 dataset consists of 60,000 32 by 32 color images in ten classes.

We trained the architecture shown in figure 5 to obtain an accuracy of 85% on the CIFAR10 dataset. The base network consists of three convolutional layers and one fully connected layer. The non-linearities are Rectified Linear Units (ReLU). The convolutional layers all use max-pooling.

Table VI shows the error rate of the extra output layers. We find similar results as before. Deeper networks are able to make more accurate classifications but every layer incurs

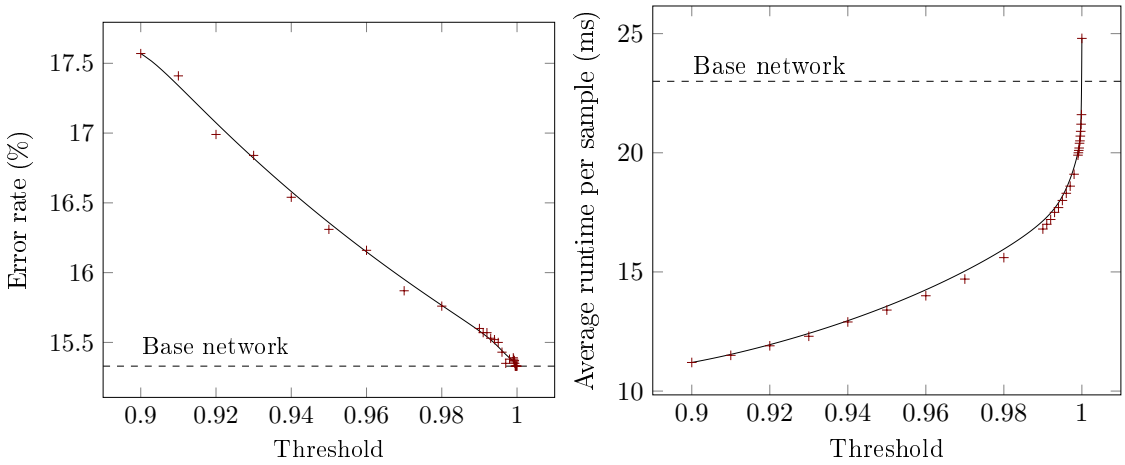


Fig. 4: Error rate (left) and runtime (right) of the cascade using varying threshold values

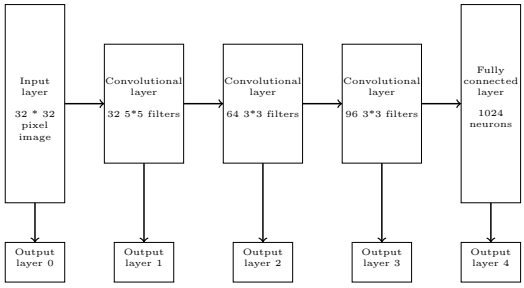


Fig. 5: 4 layer CIFAR10 network

TABLE VI: Accuracy and runtime of the network at varying depths.

Output layer	Test error rate	Average time needed to process one test sample (ms)
0	63.27%	0.2
1	33.83%	3.0
2	28.56%	6.3
3	22.87%	8.2
4	15.33%	23.0

an additional computational cost. We again used these extra trained output layers to build a cascade network. Table VII and figure 4 show the error rate and the required runtime of the cascade network using varying threshold values.

V. Threshold selection

The threshold selection is of course critical. A low threshold allows the network to return a less confident result obtained by an early layer. When a high threshold is used, most of the results will come from a deeper layer. The accuracy will be higher but so will be the computational cost. We obtained promising results on the MNIST dataset with a single shared threshold for every layer. In more complicated cases it would be better to use different threshold values for each layer. This can be done using a validation dataset, separate from the training and test set, to evaluate different choices.

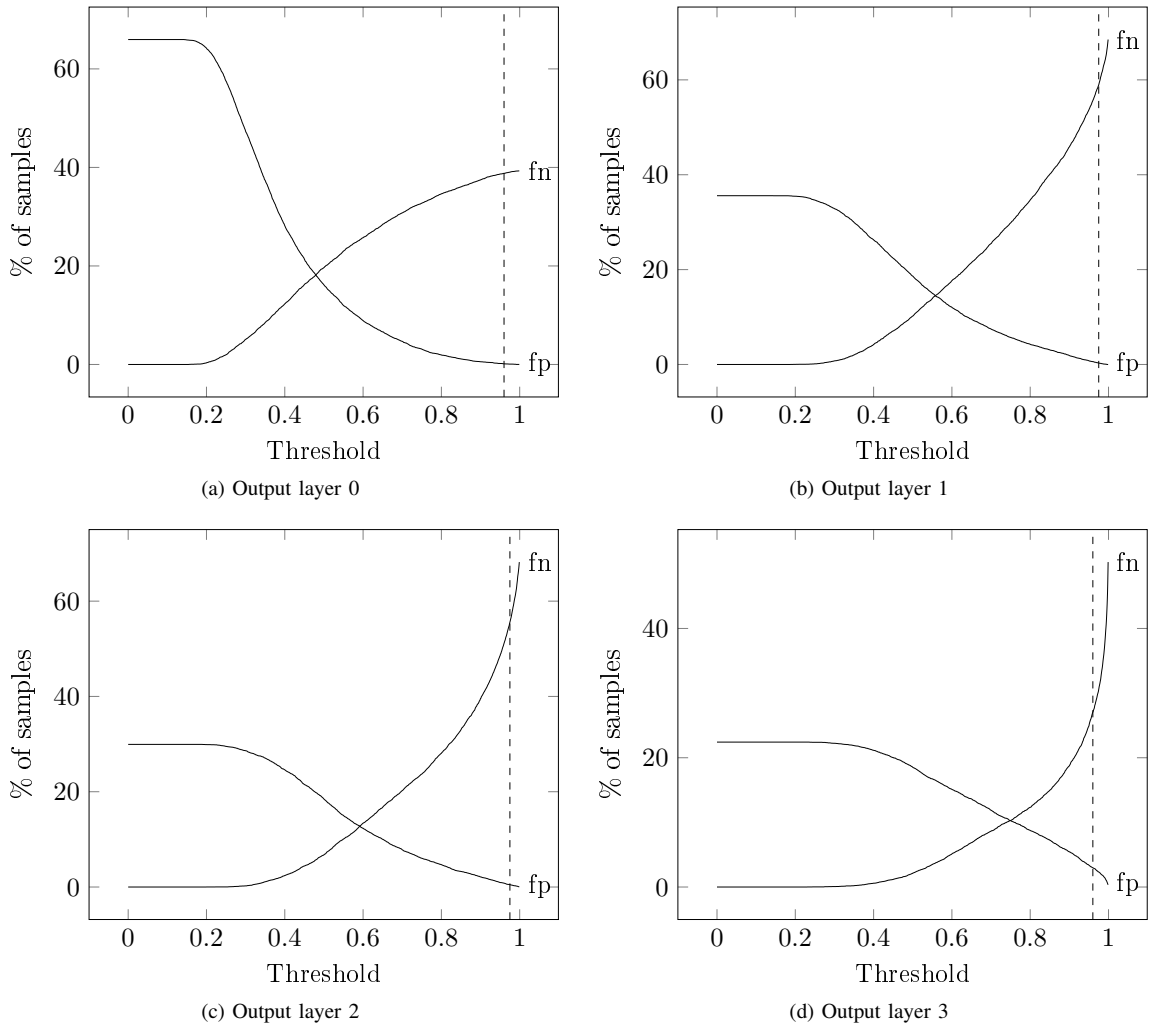
TABLE VII: Accuracy and runtime of the cascade using varying thresholds.

Threshold	Test error rate	Average time needed to process one test sample (ms)
0.9	17.57%	11.2
0.99	15.6 %	16.8
0.999	15.39%	19.9
0.9999	15.33%	21.6
1	15.33%	24.8

Every output layer returns a probability distribution over the different classes. We use the threshold to decide whether to accept or to reject the classification with the corresponding probability or confidence level. We define a false negative as a correct answer with a confidence level smaller than the threshold. A false positive is a wrongly classified sample with a confidence level higher than the threshold. A false positive directly affects the accuracy of the network: the network was confident it had a good classification but it was wrong. On the other hand, a false negative directly affects the runtime of the network. The sample was correctly classified but it was rejected so extra calculations by the deeper layers were needed.

The optimal threshold value depends on the relative importance of accuracy and speed. It is therefore interesting to investigate the number of false positive and false negative samples for a certain layer as a function of the threshold value. This allows us to see the impact of a varying threshold on both metrics. Figure 6 shows these metrics for the first four output layers of the CIFAR network. The threshold value of the last output layer is implicitly set to zero. When no other layer is able to give a confident result, the output of the last layer is accepted, regardless of its confidence level.

When the threshold value is zero, all outputs are accepted. The amount of false positives equals the error rate of the layer and there are no false negatives. When the threshold is one, no answers are accepted. There are no false positives and the amount of false negatives equals the amount of correctly classified samples. We try to minimize the threshold and the amount of false negatives while keeping the amount of false positives (the error rate) acceptable. Using this technique we



(a) Output layer 0

(b) Output layer 1

(c) Output layer 2

(d) Output layer 3

Fig. 6: False positives (fp) and false negatives (fn) of the output layers using varying thresholds. The chosen thresholds are shown by the vertical line.

decided on the threshold values 0.96, 0.99, 0.99, 0.97 (and 0). These allow an error rate of 15.8% while keeping the average time needed to process a sample limited to 15 ms.

These results are summarized in figure 7 which shows the Receiver operating characteristic (ROC) curves for the different output layers. An ROC curve is a plot of the True Positive Rate (TPR) versus the False Positive Rate (FPR) for a certain classifier. A single point represents a single measurement. The curve is obtained by varying the threshold parameters. We clearly see that the deeper layers are more accurate (they are closer to the upper left corner which represents a perfect classifier with a TPR of one and and FPR of zero).

VI. Conclusion and future work

In the future we will most likely see more practical and industrial applications of deep neural networks. While these networks are now mostly confined to high performance GPU servers, in the future they will need to run on low-end hardware.

We present an architecture for a cascade of neural network layers that allows us to stop the propagation of a sample to the deeper layers of the network when a sufficiently confident result is obtained. This is achieved by training an output layer after the input layer and after every intermediate layer. We interpret the output as confidence measures. When the confidence after a layer is larger than a given threshold, the output of that layer is used as the output of the network.

We evaluated our approach on two well-known benchmark datasets: MNIST and CIFAR10. On the MNIST dataset, the cascade network was able to achieve the same error rate as the base network (0.64%) while the required runtime was only half that of the base network. CIFAR10 represents a more difficult task. We were able to obtain a relative speed-up of 33% at an increase in error rate from 15.3% to 15.8%.

We provided an intuitive method to find suitable threshold values. In future work, we will try to optimize the thresholds or even the architecture of the network automatically given a certain required accuracy or runtime. We will also further investigate the possibility of distributing layers of a neural

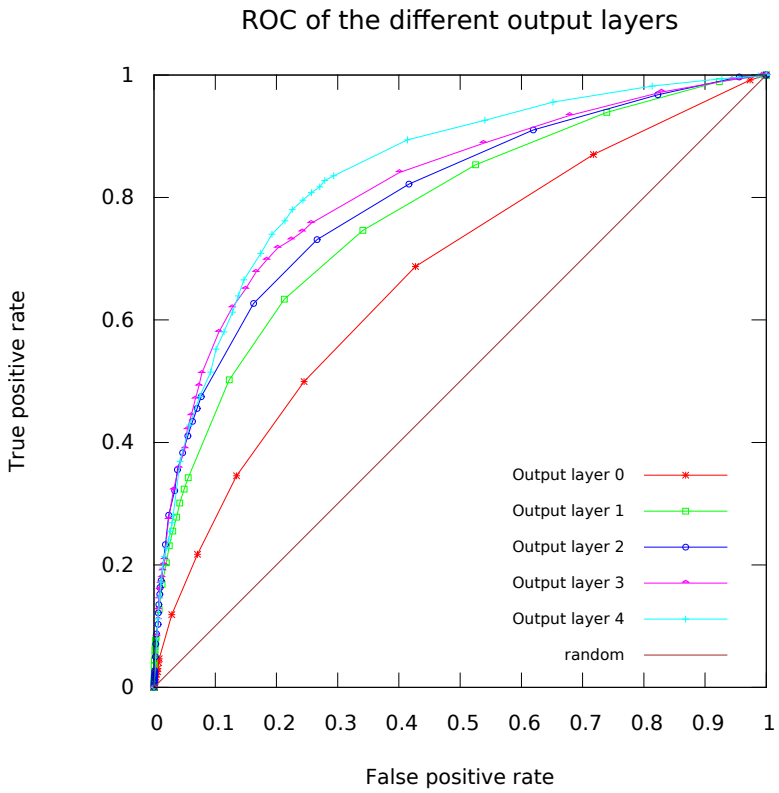


Fig. 7: Receiver operating characteristic (ROC) curves of the different output layers of the CIFAR10 network.

network over different devices.

VII. Acknowledgements

Part of this work was supported by the iMinds IoT Research Program. Steven Bohez is funded by a Ph.D. grant of the Agency for Innovation by Science and Technology in Flanders (IWT).

References

[1] D. Claudiu Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep big simple neural nets excel on handwritten digit recognition," *arXiv preprint arXiv:1003.0358*, 2010.

[2] C.-W. Tsai, C.-F. Lai, and A. V. Vasilakos, "Future internet of things: open issues and challenges," *Wireless Networks*, vol. 20, no. 8, pp. 2201–2217, 2014.

[3] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*. ACM, 2012, pp. 29–36.

[4] G. COUR, A. CET, and S. PROJE, "Neuromorphic computing gets ready for the (really) big time," *Communications of the ACM*, vol. 57, no. 6, 2014.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] R. Salakhutdinov and G. E. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *International Conference on Artificial Intelligence and Statistics*, 2007, pp. 412–419.

[7] Y. LeCun and C. Cortes, "The mnist database of handwritten digits, 1998," *Available electronically at http://yann. lecun. com/exdb/mnist*.

[8] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[9] M. Hung, M. Hu, M. Shanker, and B. Patuwo, "Estimating posterior probabilities in classification problems with neural networks," *International Journal of Computational Intelligence and Organizations*, vol. 1, no. 1, pp. 49–60, 1996.

[10] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, 1995.

[11] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

[12] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio, "Pylearn2: a machine learning research library," *arXiv preprint arXiv:1308.4214*, 2013.

[13] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[14] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *2013 12th International Conference on Document Analysis and Recognition*, vol. 2. IEEE Computer Society, 2003, pp. 958–958.

[15] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.